НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Інструментальні засоби адміністрування кафедрального хмарного сховища

Виконав: студент 4 курсу, групи ТВ-з51

Глухоманюк Володимир Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Ляшенко Максим Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент Баранюк Олександр Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2018 р.

ЗАВДАННЯ

на дипломну роботу студенту
Глухоманюку Володимиру Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи “ Інструментальні засоби адміністрування кафедрального хмарного сховища ”
керівник роботи старший выкладач Ляшенко Максим Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Microsoft Windows, мова програмування Java.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації програмного інтерфейсу, обґрунтувати обрані програмні застосунки та шляхи розробки засобу адміністрування хмарного сховища, розробити програмне забезпечення, зробити висновки за результатами роботи
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
6. Проектування архітектури програмного забезпечення
7. Визначення необхідних функцій для користувача
8. Підключення бібліотек Spring необхідних для розробки
9. Побудова REST API для роботи з функціями
Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	20.03.19-04.04.19	
2.	Розробка архітектури та загальної структури системи	04.04.19-15.04.19	
3.	Розробка структур окремих підсистем	15.04.19-25.04.19	
4.	Підготовка матеріалів	25.04.19-06.05.19	
5.	Програмна реалізація системи	06.05.19-16.05.19	
6.	Захист програмного продукту	17.05.19	
7.	Оформлення пояснювальної записки	17.05.19-30.05.19	
8.	Передзахист	31.05.19	
9.	Захист		

Студент _____ Глухоманюк В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ляшенко М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета роботи — створення інструментальних засобів адміністрування, для роботи з кафедральним хмаровим сховищем даних. В основі архітектури програмного інтерфейсу знаходиться REST. Програмний продукт створений на базі Spring Framework. Хмарове сховище розгорнуте на віддаленій машині, за допомогою сервісу Heroku.

Записка містить 67 сторінок, 5 рисунків, 2 таблиці, 3 додатків і 13 посилань.

Ключові слова: JAVA, REST API, ХМАРОВЕ СХОВИЩЕ, SPRING FRAMEWORK, HEROKU.

ABSTRACT

The purpose of the work is to create tools for administration, to work with the cathedral cloud storage. The core of the software architecture is REST. The software product is based on the Spring Framework. The cloud storage is deployed on a remote machine, with the help of Heroku service.

The note contains 67 pages, 5 figures, 2 tables, 3 annexes and 13 references.

Keywords: JAVA, REST API, KMAROV LINK, SPRING FRAMEWORK, HEROKU.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів

IDE — Integrated Development Environment.

XML — Extensible Markup Language.

API — Application Programming Interface.

GUI — Graphical User Interface.

ЕОМ — Електронна Обчислювальна Машина.

HTTP — Hyper text transfer protocol.

REST — Representation State Transfer.

TCP — Transmission Control Protocol.

ВСТУП

Для безпеки інформації та зручної взаємодії з нею було створено великі серверні центри, доступ до яких можна отримати віддалено за допомогою інтернету. Цей механізм дозволяє створювати програми для створення персонального, віддаленого сховища даних. Ситуація з доступом до необхідних даних стала набагато простішою після випуску спеціалізованих веб-сервісів для користувачів. Як і будь-які веб-сервісів, вони мають сильні і слабкі сторони, які буде розглянуто в процесі дослідження теми диплому.

Зрозуміло, що хмарове сховище є зручним інструментом для користувача, оскільки дає можливість у будь який час, з будь якого пристрою який підтримує будь який браузер отримати доступ до даних.

У даній роботі створено гнучкий програмний інтерфейс, яке дає можливість спростити розробку персонального хмарового сховища, та створено на базі програмного інтерфейсу хмарове сховище системи STAR.

Мета виконання дипломної роботи полягає у створені хмарове сховище системи STAR, та інструментів для роботи з хмаровим сховищем.

Spring Framework – це програмний каркас з відкритим кодом який використовує технологію інверсії залежності, та включаю у себе набір інструментів для розробників, такі як: Spring Core, Spring Data, Spring Security, Spring Boot. У дипломній роботі було використано Spring Core, та Spring Boot, що допомогло полегшати розгортання віддаленого серверу та написання функціоналу програмного продукту.

Програмний продукт використовує REST – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. В основі цієї архітектури закладено принципи функціонування інтернету та можливості HTTP.

Мова Java — одна з найрозповсюдженіших мов програмування. Вона є кросплатформена а саме тому може працювати на будь якій системі. Інтегроване

середовище розробки програмного забезпечення Intelij IDEA (IDE — Integrated Development Environment), яке використовується для створення й підтримки додатків, що можуть працювати під будь-якою 32-розрядною чи 64-розрядною операційною системою типу Windows, Linux або MacOS.

У першому розділі записки сформульовано постановку задачі й розглянуто проблеми щодо налагодження програмного продукту; у другому розділі проаналізовано існуючі аналоги хмарових сховищ, та сервісів які вони надають; третій розділ містить обґрунтування обраних засобів реалізації хмарового сховища; у четвертому розділі описано засоби розробки; у п’ятому розділі розглянуто програмну реалізацію інструментів взаємодії з хмаровим сховищем; шостий розділ містить опис методики використання розробленого програмного інтерфейсу.

1 ЗАДАЧА СТВОРЕННЯ ХМАРНОГО СХОВИЩА

З того часу коли комп’ютер став невід’ємною частиною нашого життя, людство почало процес відцифровування світу, інакше кажучи, все що можна перенести у комп’ютер те активно переноситься, і будь який вид даних якнайкраще підпадає під цю задачу. Існує декілька способів зберігати дані.

Даними може слугувати будь яка цифрова інформація, текстова інформація, зображення, документи, аудіо записи, відео записи, персональні дані.

Зовсім інша проблема спрощення взаємодії з цими даними. І одним із провідних рішень нашого часу є хмарні сховища.

Хмарне сховище – це онлайн-сховище, всі дані якого зберігаються на серверах. Доступ до цього сховища, все на платній, або безкоштовній основі надається в користування клієнтам. В хмарних сховищах дані зберігаються і оброблюються у так званій “хмарі”, для клієнта це виглядає як один великий віртуальний сервер, куди він може завантажити інформацію.

Отже для збереження та обробки інформації яка знаходиться у мережі STAR необхідно розробити хмарне сховище, за для збільшення швидкості роботи з файлами, безпеки передачі та збереження даних, зручності доступу та оперування даними.

1.1 Взаємодія користувача з застосунком

Застосунок або прикладна програма — комп’ютерна програма, що дає змогу розв’язувати конкретні прикладні задачі користувача. Тобто, необхідно розглянути саме взаємодію між користувачем та програмою. [1]

У переважній більшості випадків ця взаємодія полягає в отриманні даних програмою від користувача. Наприклад, програма реєстрації користувача, отримує

від нього дані, оброблює їх, зберігаючи їх у БД, таким чином користувач взаємодіє з програмою при реєстрації.

Програми можуть надавати набір функцій для взаємодії — API. Або навіть мати цілі механізми взаємодії, які називають протоколами, коли використовується безліч послідовних викликів функцій, наприклад, для складного узгодження будь-яких параметрів [2]. Великим кроком вперед є створення універсальних структур — XML і Json. За допомогою цих форматів можна передавати необхідні програмі дані за допомогою HTTP, значно полегшуючи "спілкування" користувача з системою.

Також програма може взаємодіяти не лише з користувачем, але і іншими системами. Наприклад:

— веб-сайт може використовувати реляційну СУБД, таку як MySQL, MS SQL Server або Oracle;

— програма взаємодіє з різними бібліотеками;

— програма інтегрується з іншими системами. Наприклад, CPM-система інтегрована з 1С, системами IP-телефонії, системами еквайрингу. Сайти можуть бути інтегровані з системами оплати, з соціальними мережами.

Як і будь-яка інша система, програма складається з компонентів. У програмуванні часто повторно використовують компоненти, у випадку якщо в них реалізовано необхідний функціонал. Це швидше, ніж створювати свій компонент, налагоджувати і згодом супроводжувати його. Для прикладу, відправлення листа — один компонент, запис в лог — інший компонент, побудова графіків — третій, збереження інформації про файл – четвертий. Тому багато програм використовують ті самі компоненти.

Часто програми дають можливість розширювати свою функціональність: текстові редактори і електронні таблиці дозволяють створювати макроси, браузери — додавати розширення, все CMS — доповнювати їх сторонніми компонентами і плагінами.

Сама операційна система — це програма, єдине призначення якої — існування інших програм. Вона спрощує їхнє існування, ізолює від заліза, надає можливість

використання графічного інтерфейсу і дає доступ до мережі, її мета — забезпечити роботу інших програм.

При цьому і операційні системи, і платформи, і навіть CMS у багатьох випадках набагато складніші від тієї програми, заради якої їх використовують, вони можуть включати безліч найрізноманітніших функцій, що забезпечують роботу, розділяти ресурси, надавати механізм авторизації, спрощувати звернення до бази даних. Це відбувається і за рахунок своєї універсальності (наприклад, операційна система Windows містить кілька підсистем Win32, OS/2) для забезпечення роботи для всіх можливих програм.

Універсальність завжди призводить до додаткової складності. Це стосується і тих компонент, які програмісти використовують при написанні програми, вони набагато складніші й мають набагато більший набір функцій, багато з яких не потрібні і не використовуються.

1.2 Проблем сумісності програмного забезпечення

При розробці програми яка працює з файловою системою віддаленого серверу який є “хмарою” могла би виникнути проблема сумісності операційної системи з програмою написаної на певній мові програмування. Наприклад системи Windows та Unix мають різні командні рядки, набір команд у яких також різниться. У цьому випадку було обрано мовою для розробки Java, яка окрім того що є багатоплатформовою, тобто не працює однакого на усіх системах, також має вбудований механізм роботи з файлами.

Багатоплатформовість – здатність програми працювати правильно у різних системах. У випадку з Java це досягається за допомогою того що код спочатку компілюється у проміжний код, котрий потім інтерпретується тією системою у якому знаходиться програма.

1.2.1 Відсутність необхідних ресурсів

Також, є проблема мінімальних системних вимог для роботи програми. Деякі сучасні програми потребують великою потужності системи для коректної, роботи. У випадку хмарового сховища, плюсом є те, що все що необхідно системі, так це мати достатню потужність що працювати з браузером. Зазвичай котрі не використовують багато внутрішнього ресурсу системи.

Також якщо використовувати телефон для доступу у хмарне сховище, достатньо буде лише звичайного браузера.

1.2.2 Помилки в реєстрі

Реєстр — це база даних, яка зберігає параметри і налаштування для операційних систем. Він містить інформацію і налаштування для всіх апаратних засобів, програмного забезпечення, користувачів тощо. Помилки в ньому негативно позначаються на всіх процесах, які відбуваються в комп’ютері. Причиною виникнення збоїв у реєстрі є некоректно написані програми, які прописують свої файли і посилання в різних місцях. У випадку з хмаровим сховищем це досить актуально проблема, так як користувач працюючи з програмним інтерфейсом може видаляти файли та директорії, користувач може пошкодити системні файли, або файли програми, що призведе до зупинення, або не коректної роботи системи, або програми.

Для розв’язання цієї проблем було ведено необхідні тести та перевірки, котрі обмежують область роботи користувача.

Висновки до розділу 1

У час коли кожен день створюються неймовірна кількость даних, необхідно створювати хмарні сховища, котрі будуть забезпечувати безпеку, та швидкість доступу до цих самий даних.

2 АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ ХМАРНИХ СХОВИЩ ТА СЕРВІСІВ ЯКІ ВОНИ НАДАЮТЬ

На сьогоднішній день існує багато сервісів та платформ які надають послуги віддаленого серверу та інструментів для розробника, які допоможуть при розсортуванні власного хмарового сховища. Під платформою я маю на увазі назву сукупності сервісів, наприклад: AWS (Amazon Web Services), Google Cloud: Cloud Computing Services, IBM Cloud, Heroku, і так далі. Під сервісом я маю на увазі конкретний сервіс котрий забезпечує розробника необхідним інструментом для реалізації будь якого функціоналу, наприклад: AWS EC2 – надає розробнику інструменти для обчислювані ресурси у хмаровому середовищі, Google Drive – надає розробнику масштабуєме хмарове сховище.

Кожен з сервісів має свій інтерфейс котрий описаний на офіційних сторінках платформ. Та свою інструкцію підключення до створеного розробником програмного забезпечення, та розгортання його на віддаленому сервері. Так наприклад що твоя програма працювала необхідно її завантажити на сервер, та налаштувати.

Базуючись на своєму завданні та на проаналізованих платформах та інструментах, було прийнято рішення, що для вирішення задачі буде достатньо написати власний функціональний інтерфейс, та скористатися платформою лише для отримання власного віддаленого серверу. Для цього було прийнято рішення використати платформу Heroku.

Heroku – це хмарна PaaS-платформа, що була створена у 2007 році, і я однією з перших хмарних платформ у світі. З початку існування ця платформа підтримувала лише мову програмування Ruby, але на даний момент цей список виглядає наступним чином: Java, Node.js, Scala, Clojure, Python, PHP. На серверах Heroku встановлено операційні системи Debian або Ubuntu. Також Heroku була

обрана оскільки команда що займається безпекою на функціонуванням серверів, є дуже професійною, та гарантує високу якість з'єднання та відказостійкість. [3]

2.1 Основні поняття технології Web Services

Веб-служба, або веб сервіс, який ідентифікується URI (Uniform resource identifier, компактний рядок літерал, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс) та публічні інтерфейси прив'язки якого визначені та описані мовою XML. Опис цієї програмної системи може бути знайдено іншими програмними системами, які можуть взаємодіяти з цією системою відповідно то цього опису з використанням повідомлень, що базуються на XML та передаються за допомогою інтернет протоколів. Термін "веб-служба" організацією W3C (консорціум всесвітньої павутини) застосовується до багатьох різних систем, але в основному термін стосується клієнтів та серверів, що взаємодіють за допомогою повідомлень протоколу SOAP. В обох випадках припускається, що існує також опис доступних операцій у форматі WSDL. Хоча наявність цього опису не є вимогою SOAP, а радше передумовою для автоматичного генерування коду на платформах Java та .NET на стороні клієнта. [4]

SOAP (Simple Object Access Protocol) – це протокол обміну структурованими повідомленнями у розподілених обчислювальних системах, що базується на форматі XML. [5]

XML (Extensible Markup Language) – запропонований консорціумом World Wide Web Consortium стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. [6]

WSDL (Web Services Description Language) – це мова визначення інтерфейсу веб-сервісу заснована на XML, що описує функціональність веб-сервісу і спосіб доступу до нього. [7]

Такий підхід до використання веб сервісів як SOAP є достатньо складним, та повільним за рахунок збільшення обсягу повідомлення та зменшення швидкості обробки, і підходить більше для великих систем, де опис повинен бути достатньо

гнучким, що дозволяє використовувати різні транспортні потоки, так стандартні реалізації використовують HTTP як транспортний протокол, однак також можна використовувати JMS (Java Message Services) та SMTP (Simple Mail Transfer Protocol).

Також веб сервіси використовують підхід REST. У цей підхід закладено принцип функціонування Інтернету, зокрема, можливості HTTP. Основною відмінністю від SOAP є те, що дані повинні передаватися у вигляді невеликої кількості стандартний форматів, наприклад. HTML, JSON, XML, та будь який REST протокол повинен підтримувати хешування та не повинен залежати від мережевого прошарку, не повинен зберігати інформацію про стан між парами “запит – відповідь”.

Для оперування даними, та функціями у REST використовуються HTTP запити, які відповідають операціям CRUD. Так у HTTP виділяють основні 4 запити:

— GET – отримати представлення ресурсу;

— POST – створити новий ресурс на місці даного використавши передане представлення;

— PUT – замінити стан поточного ресурсу станом, що описується переданим представленням;

— DELETE – видаляє ресурс.

Також існують інші запити, але використовуються вони не так часто, наприклад:

Запити які використовують для дослідження API:

— HEAD — отримати заголовки, які б відсилались разом з представленням цього ресурсу, але не саме представлення;

— OPTIONS — визначити список методів, на які цей ресурс відповідає. [8]

Основні недоліки та переваги веб сервісів, ми можемо описати наступним чином (таблиця 2.1).

Таблиця 2.1. Недоліки та переваги веб сервісів

Переваги	Недоліки
Веб-служби забезпечують взаємодію програмних систем незалежно від платформи	Більш низька продуктивність у порівнянні з технологіями CORBA, DCOM за рахунок використання текстових XML повідомлень
Веб-служби базуються на відкритих стандартах та протоколах. Завдяки використанню XML досягається простота розробки та відлагодження веб-служб	
Використання інтернет-протокола HTTP забезпечує взаємодію програмних систем через міжмережевий екран	

COBRA (Common Object Request Broker Architecture) – це запропонований консорціумом OMG технологічний стандарт розробки розподілених застосунків. DCOM (Distributed Component Object Model) – це власна технологія Microsoft для організації взаємодії між компонентами програмного забезпечення, розподіленого між комп’ютерами в мережі.

Таблиця 2.2. Різниця між SOAP та REST [9]

SOAP	REST
Це протокол	Це архітектурний стиль
Означає протокол доступу до об’єктів	Означає перенос репрезентативного

	стану
SOAP не може використовувати REST, тому що це протокол, а REST це архітектурний паттерн	REST може використовувати служби SOAP тому, що REST – це концепція і може використовувати будь які протоколи, такі як HTTP та SOAP
Використовує сервісні інтерфейси для розкриття бізнес логіки	Використовує уніфіковані ідентифікатори ресурсу для розкриття бізнес-логіки
Визначає суворо дотримувані стандарти	Не визначає занадто багато стандартів
Вимагає більшу пропускну здатність полоси та ресурсу	Вимагає меншу пропускну здатність полоси та ресурсу
Визначає свою власну безпеку	Web-служби RESTful наслідують заходи безпеки від основного носія
Дозволяє тільки формат XML	Дозволяє використовувати різні формати даних, такі як звичайний текст, HTML, XML, JSON, тощо.
Дані не кешуються	Дані кешуються
Подібний до десктопного додатку користувача, який приховано спілкується з сервером	Подібний до браузера та використовує стандартні методи. Додаток повинен міститися всередині нього
Повільніший	Швидший
Працює на HTTP, але огортає повідомлення в оболонку	Використовує заголовки HTTP для зберігання мета-інформації

У таблиці 2.2. відображені загальні положення що то різниці SOAP та REST.

Висновки до розділу 2

У даному розділі було описано два основних типи веб-сервісів, проаналізовані їх основні відмінності, та проведено аналіз недоліків та переваг кожного з типів. Тому базуючись на цій інформації було зроблено вибір на використані REST.

3 АНАЛІЗ І ОБҐРУНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ ХМАРНОГО СХОВИЩА ДАНИХ

У процесі створення програмного продукту важливо правильно обрати засоби реалізації хмарного сховища даних. Проаналізувавши основні помилки при створенні програмного забезпечення і підкресливши важливі фактори для створення власного API, враховувалися різні фактори. Основним завданням була економія часу на розробку, якість, надійність і швидкість роботи програмного продукту.

Аналізуючи можливі підходи для досягнення поставленої мети було розглянуто кілька варіантів створення необхідної взаємодії.

3.1 Набір бібліотек Spring Framework

Фунціонал Spring framework — це вільно поширюваний набір динамічних бібліотек. Завантажити його можна з офіційного сайту і використовувати на свій розсуд.

Для використання даного фрейморку я використав іншу технологію, яка значно спрощую процес конфігурації проекту у цілому, така технологія має назву Maven. Про технологію Maven буде йти мова у наступному підрозділі.

Spring Framework – це програмний каркас з відкритим кодом, який складається з декількох основних модулів, котрі забезпечують легший процес розробки програмного продукту, та значно збільшує надійність роботи програмного коду, та його гнучкість. Основними модулями Spring Framework є:

— Контейнер Інверсії управління: Конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через Інверсію управління

— Аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури

— Доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних

— Управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів

— Модель-Вигляд-Управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful.

— Аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security (колишня система безпеки AcerI для Spring).

— Віддалене керування: конфігураційний вплив і управління Java-об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX

— Тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів

Також Spring Framework включає у себе технологію Spring Boot, яка дозволяє спростити конфігурацію програмного продукту, а саме, підключати та виключати необхідні розробнику додатки Spring, по використовувати вбудований сервлет-контейнер, наприклад Tomcat. Сервлет це скорочена назва Java Servlet, котра має свою специфікацію, API. Java Servlet API – це стандартизований API для створення динамічного контенту до веб-сервера, використовуючи платформу Java. Сервлети — аналог технологій PHP, CGI і ASP.NET. Сервлет може зберігати інформацію між

багатьма транзакціями, використовуючи HTTP кукіз, сесії або через редагування URL.

3.2 Технологія Apache Maven

Технологія Apache Maven – засіб автоматизації роботи з програмним продуктом, який використовується в Java проектах. Основна функція цієї технології є управління та складання програми. Аналогом цієї технології є Apache Ant, котрий значно складніший для використання, аніж Maven. Maven використовує формат даних у вигляді XML. Принцип роботи Maven такий, XML-файл описує проект, його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах. Для опису програмного проекту який потрібно побудувати (build), Maven використовує конструкцію відому як Project Object Model (POM), залежності від зовнішніх модулів, компонентів та порядку побудови. Виконання певних, чітко визначених задач — таких, як компіляція коду та пакетування відбувається шляхом досягнення заздалегідь визначених цілей (targets).

Ключовою особливістю Maven є його мережева готовність (network-ready). Двигун ядра може динамічно завантажувати плагіни з репозиторію, того самого репозиторію, що забезпечує доступ до багатьох версій різних Java-проектів з відкритим кодом, від Apache та інших організацій та окремих розробників. Цей репозиторій та його реорганізований наступник, — Maven 2 репозиторій, — намагається бути де-факто механізмом для дистрибуції Java програм, але прийняття його в такій якості йде повільно.

Maven забезпечує підтримку побудови не просто перебираючи файли з цього репозиторію, але й завантажуючи назад артефакти у кінці побудови. Локальний кеш звантажених артефактів діє як первісний засіб синхронізації виходу проектів на локальній системі. [10]

3.3 Інструменти розробки хмарного сховища на базі Amazon Web Services

Для створення хмарного сховища на базі AWS, необхідно використовувати декілька сервісів надаваємих компанією. Основою для хмарного сховища є сервіс S3, який слугує базою для подальшої роботи.

Amazon Simple Storage Service (Amazon S3) - це сервіс зберігання об'єктів, що пропонує високі показники продуктивності, масштабованості, доступності та безпеки даних. Це означає, що користувачами можуть бути компанії будь-яких розмірів і з будь-яких областей діяльності. Вони можуть використовувати сервіс для зберігання і захисту будь-яких обсягів даних в різних ситуаціях, наприклад для забезпечення роботи сайтів, мобільних додатків, для резервного копіювання та відновлення, архівації, корпоративних додатків, пристроїв IoT і аналізу великих даних. Amazon S3 пропонує прості у використанні інструменти адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно потребами вашого бізнесу або законодавчими вимогами. Amazon S3 забезпечує високий рівень надійності і зберігає дані мільйонів додатків в інтересах користувача. Сервіс Amazon S3 пропонує кілька класів сховища, призначених для різних прикладів використання. До них відносяться сховище загального призначення S3 Standard для часто використовуваних даних, S3 Intelligent-Tiering для даних з невідомими або мінливими схемами доступу, S3 Standard-Infrequent Access (S3 Standard-IA), S3 One Zone-Infrequent Access (S3 One Zone-IA) для даних, що вимагають тривалого зберігання, але менш частого доступу, Amazon S3 Glacier (S3 Glacier) і Amazon S3 Glacier Deep Archive (S3 Glacier Deep Archive) для довгострокового зберігання та цифрової архівації даних. Amazon S3 також надає можливості управління даними протягом їх життєвого циклу. Після налаштування політики життєвого циклу S3 дані будуть автоматично переміщатися в інший клас сховища, при цьому змінювати додаток, або веб-аплікацію не потрібно.

Також для модифікації створеного хмарного сховища на базі AWS S3 існує багато різних сервісів, платних та без коштовних. Наприклад, декілька основних

сервісів:

— Alexa for Business Розширення можливостей організації за допомогою Alexa
— Amazon WorkMail Захищена електронна пошта та календарі
— Amazon EC2 Віртуальні сервери в хмарі
— Amazon EC2 Auto Scaling Масштабування обчислювальних ресурсів в залежності від навантаження
— Amazon Elastic Container Registry Зберігання та витяг образів Docker
— Amazon Elastic Container Service Запуск контейнерів Docker і управління ними
— Amazon Lightsail Запуск приватних віртуальних серверів і керування ними
— AWS Batch Запуск пакетних завдань в будь-якому масштабі
— AWS Elastic Beanstalk Запуск інтернет-додатків і управління ними
— AWS Fargate Запуск контейнерів без необхідності керувати серверами або кластерами
— AWS Lambda Запуск програмного коду без турбот про серверах
— AWS Outposts Запуск інфраструктури AWS в локальному середовищі
— AWS Serverless Application Repository Пошук, розгортання і публікація бессерверной додатків
— VMware Cloud on AWS Створення гібридного хмари без спеціального обладнання
— Amazon Connect Сервіс хмарного контакт-центру
— Amazon Pinpoint Залучення користувачів по всіх каналах з можливістю індивідуального налаштування
— Amazon Simple Email Service (SES) Отримання і відправка електронної пошти
— Amazon Aurora Керована реляційна база даних з високою продуктивністю
— Amazon DynamoDB Керована база даних NoSQL
— Amazon DocumentDB (підтримує сумісність з MongoDB) Цілком керована документная база даних
— Amazon ElastiCache Система кешування в пам'яті
— Amazon Neptune Повністю керований сервіс графових баз даних

— Amazon Quantum Ledger Database (QLDB) Повністю керована база даних для реєстрів
— Amazon RDS Керований сервіс реляційних БД для MySQL, PostgreSQL, Oracle, SQL Server і MariaDB
— Amazon RDS on VMware Автоматизація управління базою даних в локальному середовищі
— Amazon Redshift Швидке, просте і економічне зберігання даних
— Amazon Timestream Повністю керована база даних часових рядів

Отже висновок наступний. AWS, є чудовим виробом як платформа для створення хмарного середовища, хмарних систем різного розміру та функціоналу але основним недоліком, який став вирішальним фактором, це те, що основний код усіх сервісів не надає можливості розробнику модифікувати його, а саме розробник стає прив'язаний до оновлень AWS так не має жодного впливу на цей процес. [11]

3.4 Інструменти розробки хмарного сховища на базі Google Cloud Platform

Google Cloud Platrorm - це платформа виду «інфраструктура як сервіс» (IaaS), що дозволяє клієнтам створювати, тестувати і розгортати власні додатки на інфраструктурі Google, в наданих віртуальних машинах. Google Compute Engine надає віртуальні машини, що працюють в інноваційних центрах обробки даних Google і всесвітньої мережі. Подібно до інших платформ які надають сервіси для розробки хмарних середовищ, Google Cloud обіцяє безпечну та швидкісну роботу з будь якими об'ємами даних [12]. Також google надає велику кількість додаткових сервісів для роботи та масштабування персонального хмарного середовища, що дозволяє розробнику створювати саме таку систему, яка йому потрібна, без зайвих функцій, які лише будуть заважати користувачу.

Для початку роботи в Google Cloud Platform, необхідно створити обліковий запис у Google та перейти на сторінку Google Cloud Platform, та вибрати модель

роботи, а саме, користувач може обрати платну чи безплатну версію хмарного середовища. В залежності від обраного, розробнику буде надано певну конфігурацію віддаленого серверу, та переліків сервісів які можна використовувати. Також розробник може додаткового підключити безкоштовні та платні сервіси до створеного середовища.

Перевагами цього середовища є:

- Масштабуємість створеного хмарного середовища
- Надані інструменти адміністрування хмарного середовища
- Інтерфейс для роботи з стабільністю хмарного сховища
- Велика екосистема розробки

Основними недоліками для прі роботі з хмарним середовищем є інколи низька швидкість операцій з файловою системою із за особливості наданих розробникам сервісів. А також велика ціна на обслуговування хмарного сховища та надання сервісів для його масштабування.

3.5 Інструменти розробки хмарного сховища на базі Heroku

Heroku — хмарна PaaS-платформа, що підтримує ряд мов програмування. Компанією Heroku володіє Salesforce.com. Heroku, одна з перших хмарних платформ, з'явилась в червні 2007 року і спочатку підтримувала тільки мову програмування Ruby, але на даний момент список підтримуваних мов також включає в себе Java, Node.js, Scala, Clojure, Python і PHP. На серверах Heroku використовуються операційні системи Debian або Ubuntu (яка також заснована на Debian) [3]. Heroku, на відміну від інших аналогів, надає лише віддалені сервера, але як альтернатива веб сервісам, які надають велика кількість конкурентів, Heroku пропоную використовувати певну площадку, на якій розробники зі всього світу зможуть публікувати свої персональні додатки, при чому замість розробників можуть знаходитись цілі компанії, які створюють такі додатки, та публікують їх. Додатки можуть бути як платні так і безкоштовні, в основному це залежить від

внутрішнього функціоналу. Це дозволяє значно розширити додатковий функціонал для свого хмарного сховища, та не бути прив’язаним до оновлень однієї компанії.

Зазвичай при розробці хмарного сховища значною мірою акцентується увага на виборі мови розробки, тому Heroku надає підтримку багатьох мов розробки, це дозволяє не зациклюватись на цьому питані, та значно полегшати розробку багатьох систем, адже багато з цих мов можуть використовувати так звану нативного коду, що дозволяє взаємодію декількох мов одночасно у єдиному середовищі.

Heroku надає як платні конфігурації серверів, так і безкоштовні, що дозволяє вибрати машину під характеристику своєї системи. А також надає достатньо легкий шлях інсталяції власного додатку на сервер, та інтерфейс взаємодії з сервером, це дозволяє користуватися сервером через командну строку, це значно спрощую адміністрування системи, та її тестування.

Heroku має сервери у будь-який, зручній точці світу, це дозволяє добитися швидкої реакції сервера на запит користувача, а також за допомогою того, що на сервери встановлена система Ubuntu, це дозволяє працювати хмарному середовищу стабільно, так не забирає у системи великих ресурсів на роботу.

Висновки з приводу цього наступні, із-за простоти налаштування та великої бібліотеки додатків створених іншими розробниками, для реалізації хмарного середовища було обрано саме Heroku. Велика надійність серверів та розташування серверу забезпечують чітку на безпечну роботу хмарного сховища даних, що є необхідною умовою для створення персональної системи. А також роботи підтримки мови програмування Java стала вирішальним пунктом у виборі системи.

Висновки до розділу 3

У даному розділі описуються основні засоби створення інструментів адміністрування хмарного середовища. Розглядається технології необхідні, для розробки гнучкого інтерфейсу, та середових які зможуть задовольнити вимогам швидкості, безпеки та невідказності розробленого хмарного сховища. Також

обґрунтовуються переваги обраних інструментів та платформ для реалізації взаємодії між засобами розробки.

4 ЗАСОБИ РОЗРОБКИ

Враховуючи необхідність використання сучасної мови програмування Java, було використано Framework Spring Boot, що забезпечить більш зручну реалізацію поставленого завдання.

Для розробки функціональної частини використовувалися базові функції роботи з файловою системою серверу за допомогою пакету Java NIO2.

4.1 Середовище розробки Intellij IDEA 2019

Середовище розробки Intellij IDEA 2019 [13] — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows. Також до складу IDE входить набір інструментів для створення програмного забезпечення які спрощують планування та розробки призначеного для розробника інтерфейсу взаємодії, а також тестування, налагодження, аналіз якості коду й продуктивності, розгортання в середовищах клієнтів і збору даних. Ці інструменти призначені для максимально ефективної спільної роботи, всі вони доступні в інтегрованому середовищі розробки.

Середовище розробки Intellij IDEA можна використовувати для створення різних типів додатків — від простих додатків для магазину та ігор для мобільних клієнтів до великих і складних систем, які обслуговують підприємства й центри обробки даних. У середовищі можна створювати:

— додатки та ігри, які виконуються на будь якій платформі, навіть мобільній;

— веб-сайти і веб-служби на основі Spring Framework, JQuery, AngularJS і інших популярних платформ;

— ігри і графічні додатки для різних пристроїв

За замовчуванням Intellij IDEA забезпечує підтримку Java, Python, Scala, PHP. Середовище розробки Intellij IDEA добре працює й інтегрується зі сторонніми додатками, наприклад, SQL Workbench і Apache Maven за допомогою розширень. Користувач також може самостійно розширити Intellij IDEA, створивши власні інструменти для виконання спеціалізованих завдань.

Є можливість створювати додатки для кількох пристроїв, персональних комп'ютерів і веб-додатки, використовуючи хмарні ресурси, або використовуючи вже розроблені додатки. Міжплатформні технології допомагають створювати додатки для Windows, MacOS, Ubuntu, Android і iOS за допомогою одного проекту для розробки.

Також Intellij IDEA пропонує потужні функції, які допомагають швидко розібратися в програмному коді. Також функція VCS (Version Control System) дозволяє відслідковувати зміни у загальному проекті, котрі внесли інші розробники, та налаштувати розробку будь якого програмного забезпечення цілою командою людей.

4.2 Мова програмування Java

Для реалізації програми було використано інтегроване середовище програмування Intellij IDEA, що безкоштовно поширюється компанією JetBrains за ліцензією для студентів, і мову програмування Java, можливості якої описано нижче.

Основною мовою розробки програм на платформі Spring Framework є мова Java. У мові вдало поєднуються випробувані засоби програмування з останніми нововведеннями і надається можливість для ефективного і дуже практичного

написання програм, призначених для обчислювального середовища сучасних підприємств.

Java дає можливість створювати кросплатформові програми що дозволяє використовувати розроблений продукт на будь якій платформі яка має встановлену необхідну версію Java, тобто розробник пише однаковий код для усіх операційних систем одночасно. Це дуже зручно використовувати, коли необхідно реалізувати взаємодію з програмою, що має використовуватись на різних операційних системах, або серверах.

Технологія роботи в цьому IDE базується на ідеях об'єктно-орієнтованого програмування. В основі об'єктно-орієнтованого програмування (ООП) лежить ідея об'єднання в одній структурі даних і дій, які виконуються над цими даними.

Процес написання веб аплікації розділяється на дві частини: перша частина — розробник створює функціональний інтерфейс, який використовуватиметься для взаємодії користувача з функціями.

Друга частина — за допомогою будь якого фреймворку візуалізації створює графічний інтерфейс користувача встановлює потрібні розміри, налаштовує властивості.

Середовище є одним з найбільш визнаних і популярних IDE для швидкої розробки веб систем.

4.3 Набір функцій NIO2

NIO2 – це вбудований набір функцій у стандартний функціонал мови Java, починаючи з 6-ї весії. У старій весії NIO було модифікавано та додано наступні властивості:

Path новий фундаментальний підхід для роботи з файлами і директоріями

Клас File инкапсулирует основні допоміжні методи для роботи з файлами

Модерн новий підхід у вирішенні класичних завдань управління введення виведення IO. Введення в асинхронне IO API more New I / O ще новіше IO - new IO вже було присутнє в java 1.6 у версії 1.7 апі набуло ще більше нововведень. У версії

8 зміни вже не значні і базових речей не торкнулися. За посиланням можна подивитися яку користь вводили від версії до версії. Все, що ставитися до NIO2 (всі класи) живуть в пакеті java.nio.

Це абсолютно новий підхід, на відміну від старого java.io.File покликаний повністю його замінити у всіх аспектах, що стосуються взаємодії з файлової системою.

NIO2 вже по-замовчуванню має зручні можливості виконання в багатопотоковому додатку. Без болісних змін можна виконувати операції роботи з фалового системою або мережею в фоновому потоці.

З усіх боків спрощує кодинг і привносить навіть нові можливості? які раніше були недоступні. Наприклад робота безпосередньо з символічними посиланнями. Ось деякі з методів, які сильно спрощують життя:

Files.walkFileTree()

Files.isSymbolicLink()

Files.readAttributes()

Нове апі добре оптимізували для роботи з конкретною ОС використовує її ті чи інші нативні переваги для швидкості роботи з ФС мережею, великими файлами. Покращення стосуються оптимізації роботи програми на декількох багатоядерних процесорів, які дозволяють програмі виконуватися в "справжньої" мультіпоточной манері. Являє програмістам досить просту абстракцію для мультіпоточной роботи з файлами і сокетами. Розглянемо більш бодробно можливості апі. Як я вже вказав раніше клас Path це фундамент для роботи з файловим введенням висновком не залежно від операційної системи. І може представлятися наприклад таким видом "c:/user/dir" або "/ user / dir" (* nix os). Path це абстрактна конструкція. Об'єкт можна створити і навіть працювати з ним, але він може фізично не ставитися не до одного файлу в сісетеме, поки ми його не створимо очевидним чином - Files.createFile (Path target). Інакше якщо будемо писати або читати до створення то отримаємо IOException. Теж саме трапитися якщо файл не існує, а я його спробую прорахувати. До речі говолря, JVM прив'язує об'єкти до фізичних тільки в Рантайм. Апі nio2 розділяє розташування (Path) і маніпуляцію з файл системою (клас File) Path НЕ

обмежується поданням класичних ФС. Клас так само може представляти місцезнаходження файлів і папок в архівах наприклад jar або zip. Опис основних ключових класів: Path включає методи для отримання інформації про шляхи до розташування файли або папки. Для доступу до елементів шляху. Для конвертації в одному форми. Або вилучення частини шляху. Ще є методи для розпізнання рядки., Що становить шлях і для видалення ібиточності з шляху (багатослівність). Paths клас помічник включає допоміжні методи для формування шляху наприклад, get (String first, String ... more) або get (URI uri). FileSystem клас інтерфейс до файлової системи.

FileSystems допоміжний клас з хелперами.

Створення:

Path:

Path listing = Paths.get ("/ usr / bin / zip");

або еквівалент:

Path listing = FileSystems.getDefault (). GetPath ("/ usr / bin / zip");

Отримання пов'язаної інформації:

Файл :

listing.getFileName () = zip

Кількість іменованих елементів в дорозі listing.getNameCount () = 3

Батьківський каталог listing.getParent () = bin

Корінь ФС listing.getRoot () = /

Кількість підкаталогів від кореня до файлу (глибина) listing.subpath (0, 2) = 2

Оптимізація шляхів: Можна навести посилання до реального файло на який вона посилається так

Path realPath = Paths.get ("/ usr / logs / log1.txt"). ToRealPath ();

Можна з'єднати дві папки:

Path prefix = Paths.get ("/ uat /");

Path completePath = prefix.resolve ("conf / application.properties");

Немає ніяких проблем при роботі з легаси кодом їх старого пакету java.io.File який природно в 7 Джаві теж є для зворотної сумісності.

Для цього є такі методи:

toPath () і toFile (), де перший переводить до сучасного Path, а другий до старого File.

Робота з файловою системою з точки зору апі файли і папки розглядаються як єдині суті різняться лише відповідними атрибутами. nio2 надає слудующие як стандартні, так і специфічні цього апі возмозможності Створення та видалення файлів

Path target = Paths.get ("D: \\ Backup \\ MyStuff.txt");

Path file = Files.createFile (target);

можна задати атрибут FileAttribute в прикладі на читання і запис всім. для posix систем (наприклад UNIX частково Windows ітд)

Path target = Paths.get ("D: \\ Backup \\ MyStuff.txt");

Set <PosixFilePermission> perms = PosixFilePermissions.fromString ("rw-rw-rw-");

FileAttribute <Set <PosixFilePermission >> attr = PosixFilePermissions.asFileAttribute (perms);

Files.createFile (target, attr);

Для видалення файлів потрібно мати необхідні привілеї для користувача з під якого запущений процес

Path target = Paths.get ("D: \\ Backup \\ MyStuff.txt"); Files.delete (target);

Копіювання і переміщення:

Використовуючи допоміжні методи з класу File можна копіювати і переміщати файли з різними варіантами опцій.

Path source = Paths.get ("C: \\ My Documents \\ Stuff.txt");

Path target = Paths.get ("D: \\ Backup \\ MyStuff.txt");

Files.copy (source, target, REPLACE_EXISTING);

Є багато різних опцій. Наприклад ATOMIC_MOVE - операція переміщення завершується успіхом якщо обидві сторони процесу успішні (видалено вставлено в нове місце).

Path source = Paths.get ("C: \\ My Documents \\ Stuff.txt");

Path target = Paths.get ("D: \\ Backup \\ MyStuff.txt");

Files.move (source, target, REPLACE_EXISTING, COPY_ATTRIBUTES);

Читання і запис атрибутів

```
Path zip = Paths.get ( "/ usr / bin / zip");
Files.getLastModifiedTime (zip);
Files.size (zip);
Files.isSymbolicLink (zip);
Files.isDirectory (zip);
```

Це стандартні атрибути підтримувані більшістю файлових систем, так само можна читати атрибути властиві певним ФС наприклад posix:

```
Path profile = Paths.get ( "/ user / Admin / .profile");
PosixFileAttributes attrs = Files.readAttributes (profile, PosixFileAttributes.class);
Set <PosixFilePermission> posixPermissions = attrs.permissions ();
posixPermissions.clear ();
posixPermissions.add (GROUP_READ);
...
Files.setPosixFilePermissions (profile, posixPermissions);
```

Читання і запис в файл Апі дозволяє ісать і читати файли відкриваючи їх безпосередньо для буферезірованних reader writer і читати їх за рядком. Для зворотну сумісність Path так само працює зі звичайними (старими) input output потоками.

Читання:

```
Path logFile = Paths.get ( "/ tmp / app.log");
try (BufferedReader reader = Files.newBufferedReader (logFile,
StandardCharsets.UTF_8)) {
String line;
while ((line = reader.readLine ())! = null) {
...
}
}
```

Запис:

```
Path logFile = Paths.get ( "/ tmp / app.log");
try (BufferedWriter writer = Files.newBufferedWrite (logFile, StandardCharsets.UTF_8,
```

```
StandardOpenOption.WRITE)){
writer.write ( "Hello World!");
 ..
}
```

StandardOpenOption.WRITE це один з можливих аргументів для відкриття файлу.

Аргумент типу varargs.

Для забезпечення сумісності зі старим апі треба використовувати методи обгортки для nio в класі Files:

Files.newInputStream (Path, OpenOption ...)

Files.newOutputStream (Path, OpenOption ...)

У деяких випадках для зручності читання можна скористатися додатковими методами:

```
Path logFile = Paths.get ( "/ tmp / app.log");
List <String> lines = Files.readAllLines (logFile, StandardCharsets.UTF_8);
byte [] bytes = Files.readAllBytes (logFile);
```

Сервіс відстеження змін WatchService

Для зручного моніторингу зміни файлів і атрибутів доданий новий клас java.nio.file.WatchService.

```
WatchService watcher = FileSystems.getDefault (). NewWatchService ();
Path dir = FileSystems.getDefault (). GetPath ( "/ usr / karianna");
WatchKey key = dir.register (watcher, ENTRY_MODIFY);
while (! shutdown) {key = watcher.take ();
for (WatchEvent <?> event: key.pollEvents ()) {
    if (event.kind () == ENTRY_MODIFY) {
      System.out.println ( "Home dir changed!");      }
}
key.reset ();
}
```

Код працює в нескінченному циклі (прапор shutdown). watcher.take () беремо знімок минулих подій (перший раз звичайно змін не буде тому для постійного

моніторингу цикл і потрібен) проходимся по колекції шукаємо цікавить нас - якщо є? значить сталося і було зареєстровано. key.reset (); скидаємо стан. Один з варіантів рефакторінга реалізувати патерн observer. Об'єкти зможуть працювати за методом підписки на подію.

SeekableByteChannel покращений bytechannel

У java 7 вводиться новий інтерфейс SeekableByteChannel надає зручну можливість розробнику зміни позиції курсора (відстеження) і розміру установки розміру буфера для зчитування даних з каналу

```
Path logFile = Paths.get ( "c: \\ temp.log");
ByteBuffer buffer = ByteBuffer.allocate (1024);
FileChannel channel = FileChannel.open (logFile, StandardOpenOption.READ);
channel.read (buffer, channel.size () - 1000);
```

Також NIO2 працює однакого добре на усіх операційних системах, оскільки для роботи Java використовується проміжний, байт код. Також необхідно брати до уваги, особливості файлового шляху у операційній системі, де встановлено веб-аплікацію, із застосуванням NIO2. [14]

4.4 Операційна система Ubuntu

Ubuntu — операційна система для робочих станцій, лептопів і серверів, найпопулярніший у світі дистрибутив Linux. Серед основних цілей Ubuntu — надання сучасного й водночас стабільного програмного забезпечення для пересічного користувача із сильним акцентом на простоту встановлення та користування.

Ubuntu надає користувачу мінімальний набір програм загального призначення: багатовіконне стільничне середовище, засоби для перегляду Інтернету, організації електронної пошти, офісні програми з можливістю читати і записувати файли у форматах, що використовуються в пакеті програм Microsoft Office, редактор зображень, програвач компакт-дисків тощо. Спеціалізоване програмне забезпечення, потрібне досвідченішим користувачам, можна отримати з відповідних репозиторіїв.

Серверний варіант системи включає також засоби, потрібні для організації сервера баз даних, веб-сервера, сервера електронної пошти тощо.

Ubuntu побудований на основі Debian GNU/Linux — іншого популярного дистрибутиву Лінукс. Спонсорується Canonical Ltd., власником якої є бізнесмен із ПАР Марк Шаттлворт. Назва дистрибутиву походить від зулуської концепції «убунту», яку можна висловити приблизно, як «людяність». Дистрибутив так названий для популяризації духу цієї філософії у світі програмного забезпечення. Ubuntu належить до вільного програмного забезпечення і може безкоштовно передаватись необмеженій кількості користувачів.

Офіційні підпроекти Lubuntu та Xubuntu використовують як основні стільничні середовища LXDE та Xfce відповідно замість Unity, що встановлюється за замовчуванням в дистрибутивах основного проекту. Ці дистрибутиви мають менші системні вимоги і підходять для використання на слабких та старих комп'ютерах.

До 6 лютого 2012-го ще одним офіційним підпроектом була Kubuntu із основною стільницею KDE. Canonical припинила її підтримку, пояснивши це відсутністю комерційного успіху проекту протягом 7 років.

Інший офіційний підпроект, Edubuntu, розробляється для шкіл і використання дітьми вдома.

Gobuntu є офіційним підпроектом, в якому використовується винятково вільне програмне забезпечення.

Ще одним офіційним підпроектом є JeOS (вимовляється «Джюс»), створений для використання на віртуальних машинах. Нові версії Ubuntu виходять кожні шість місяців. Кожен реліз підтримується оновленнями програмного забезпечення протягом 18 місяців. Кожні два роки виходять LTS-випуски, які підтримуються оновленнями протягом п'яти років, як десктопні версії, так і серверні (до версії 12.04

LTS-релізи для десктопів підтримувалися протягом трьох років, для серверів — п'яти). LTS релізами були версії 6.04, 8.04, 10.04, 12.04, 14.04. Останній LTS випуск, Ubuntu 16.04.1 (Xenial Xerus), вийшов 21 липня 2016. Ubuntu поставляється з найсвіжішою версією GNOME, але починаючи з версії 11.04 компанія Canonical вирішила розробити свою власну стільницю під назвою Unity. Програмне забезпечення підбирається таким чином, щоб інсталяційні пакунки та Live CD помістилися на одному компакт диску, водночас надавши користувачеві змогу створити для себе зручне робоче середовище. Live CD включає в себе версії популярних крос-платформених програм (Mozilla Firefox, Mozilla Thunderbird, Empathy, OpenOffice.org/LibreOffice та F-Spot), які дають змогу користувачеві ознайомитися із системою ще до встановлення на твердий диск. Педро Корте-Реаль (Pedro Côrte-Real) провів дослідження складу базового сховища «main» дистрибутиву Ubuntu 11.04 за обсягом коду (у вибірці фігурувало число рядків коду). З великих проектів відзначені: Linux-ядро (9%, але з них 3% займають пов'язані з ядром зовнішні компоненти, такі як iptables і udev), інструменти GNU, такі як компілятор GCC і системна бібліотека Glibc (8%), X.Org (3%), Java (6%), Mozilla (6%), KDE (8%), GNOME (5%). Але домінують в дистрибутиві маленькі проекти, сукупна частка яких становить 56%.[15]

Висновки до розділу 4

В даному розділі описано засоби, використані при розробці програмних додатків. Розроблені додатки призначені для роботи в будь якій операційній системі, тому вибір відповідного середовища був зав’язаний на більш зручній на гнучкій функціональності для роботи з мовою Java.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНСТРУМЕНТІВ АДМІНІСТРУВАННЯ КАФЕДРАЛЬНОГО ХМАРНОГО СХОВИЩА

В основі програмної реалізації інструментів адміністрування хмарного сховища лежить мова Java та архітектура розробки програмного забезпечення REST. Для створення інтерфейсу взаємодії користувача з функціями використовуються створені так звані endpoints. Користувач може викликати певну функцію за допомогою url, у яку він передає назву того endpoint який йому необхідно викликати у даний момент. Такий підхід значно спрощую розробку інструментів адміністрування, адже є відносно легким у реалізації.

5.1 Створення інтерфейсу взаємодії користувача з хмарним сховищем

Для створення інтерфейсу необхідно підключення Spring Framework, та пакету які містять анотації для створення endpoints, для цього необхідно виконати наступні дії:

— додати до Apache Maven залежність для Spring Framework Рисунок 5.1 ;
— виконати підключення до необхідного класу пакету, який містить опис необхідних для розробки анотацій ;
— створити на базі необхідного класу контролер, та заанатувати методи які буде викликати користувач.

На рисунку 5.1 відображається спосіб підключення фреймворку Spring за допомогою технології Maven.

Рисунок 5.1 – Вікно додавання Spring Framework та Junit тестів до Maven

Назва проекту визначає назву пакету, які використовуються в подальшому в проекті у різноманітних класах. Проект складається з класів, та контролерів які в свою чергу включають анотації spring.

Розглянемо типи класів які використовуються у процесі написання програмної частини. Усього використовуються такі типи класів: entity class, controller class, simple class, singleton class.

Entity class складається лише з полів, сеттерів, геттерів та конструктора ініціалізації. Ці класи призначені для репрезентації вигляду даних та зберігання інформації в БД, за допомогою анотацій які використовуються у hibernate.

Controller class як і звичайний клас складається з функцій, але він проанатований відповідно до архітектури REST, необхідні методи відмічені як endpoint.

Simple class це звичайний клас Java, який слугує для описання функціональною частини веб-аплікації.

Singletone Class це клас який не відрізняється нічим від звичайного класу, але можна створювати лише один об’єкт цього класу, таким чином забезпечити безпеку від фатальних помилок при роботі з файловою системою.

5.2 Створення основних методів взаємодії користувача з хмарним сховищем

Методи взаємодії знаходяться у класі контролері, який має назву .

За допомогою методів цього класу, користувач буде тригерети певні методи у коді, котрі після відпрацьовування будуть повертати результат роботи користувачеві.

На рисунку 5.2 подано приклад анотації класу контролера.

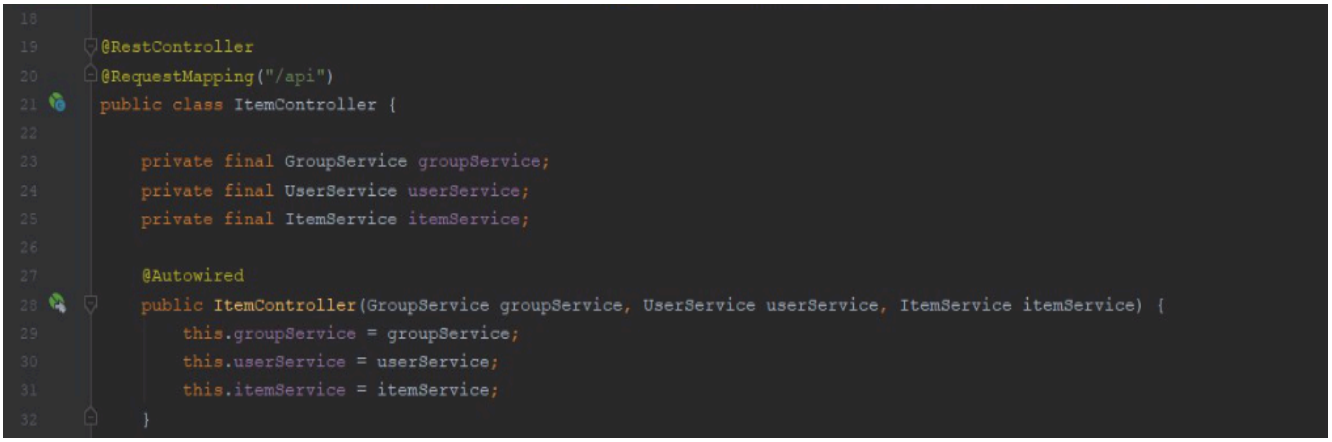

Рисунок 5.2 – приклад анотації класу контролера

Після того як користувач на звернувся до сервера з відповідним запитом, та параметрами у цьому запиті, сервер передає ці дані до відповідного endpoint, після чого створюється новий об’єкт класу singleton, якщо об’єкт ще не був створений, та викликає вже існуючий, якщо об’єкт був створений. Далі endpoint звертається до необхідного йому методу у створеному об’єкті класу, та викликає його, передаюча необхідні йому параметри.

Таким чином було створено 6 основних методів взаємодії користувача з хмарним сховищем:

— getItemById це метод який передає користувачеві папку або файл, приймає як параметр id, де id – це унікальне ім’я каталогу, де знаходиться файл або каталог.

— getAllItems це метод який передає користувачеві список усіх папок та файлів які знаходяться у каталозі

— uploadItem це метод який завантажує на сервер файл, як параметр приймає файл який необхідно завантажити

— downloadItem це метод який закачує з сервера файл, як параметр приймає id файлу
— createItem це метод який створює нову дерикторію, як параметр приймає ім’я нової директорії
— deleteItem це метод який видаляє директорію або файл, як параметр приймає id файлу або директорії
— shareItem це метод який дозволяє поширити доступ до файлу користувачеві, як параметр приймає id, унікальне імя файлу, email користувача хмарного сховища, itemPermission, параметр дозволу на передачу файлу
— shareItemWithGroup це метод який дозволяє поширити доступ до файлу групі користувачів, як параметр приймає id, унікальне імя файлу, idOfGroup унікальне ім’я групи користувачів, itemPermission, параметр дозволу на передачу файлу

Ці базові методи дозволяють створити власне хмарове середовище. Доступ до кожного з цих методів користувач може отримати за конкретно зарезервованим посиланням для певного методу. Повний перелік посилань:

— /get/{id}
— /getAllItems
— /upload
— /download/{id}
— /createItem
— /deleteItem/{id}
— /share/{id}
— /share_group/{id}

Ці посилання є відповідними endpoint для відповідних методів описаних вище.

Висновки до розділу 5

У розділі наведено детальний опис створення інструментів адміністрування хмарного сховища. Описана технологія взаємодії користувача з хмарним сховищем та приведений список базових операцій.

6 МЕТОДИКА ВИКОРИСТАННЯ РОЗРОБЛЕНОЇ ТЕХНОЛОГІЇ

Для забезпечення надійної роботи з інструментами адміністрування необхідно дотримуватись певних правил та вимог.

6.1 Вимоги до середовищ програмування

Розроблений інтерфейс не прив’язана до операційної системи адже написана на мові програмування Java, єдиною умовою до системи, є встановлена відповідна версія Java.

При розробці інтерфейсу використовувалась мова програмування Java з використанням середовища:

1. Середовище розробки Intellij IDEA 2019.

6.2 Загальний опис хмарного середовища

Розроблене хмарне середовище призначене для збереження даних, та створення ієрархії каталогів, з цілю безпечного і зручного доступу до даних.

Після переходу за посиланням: https://star-file-system.herokuapp.com/main, хмарного середовища, користувачеві необхідно буде пройти перевірку, авторизацію рисунку 6.1.

Для отримання доступу до хмарного сховища необхідно мати виданий необхідний доступ адміністратором системи.

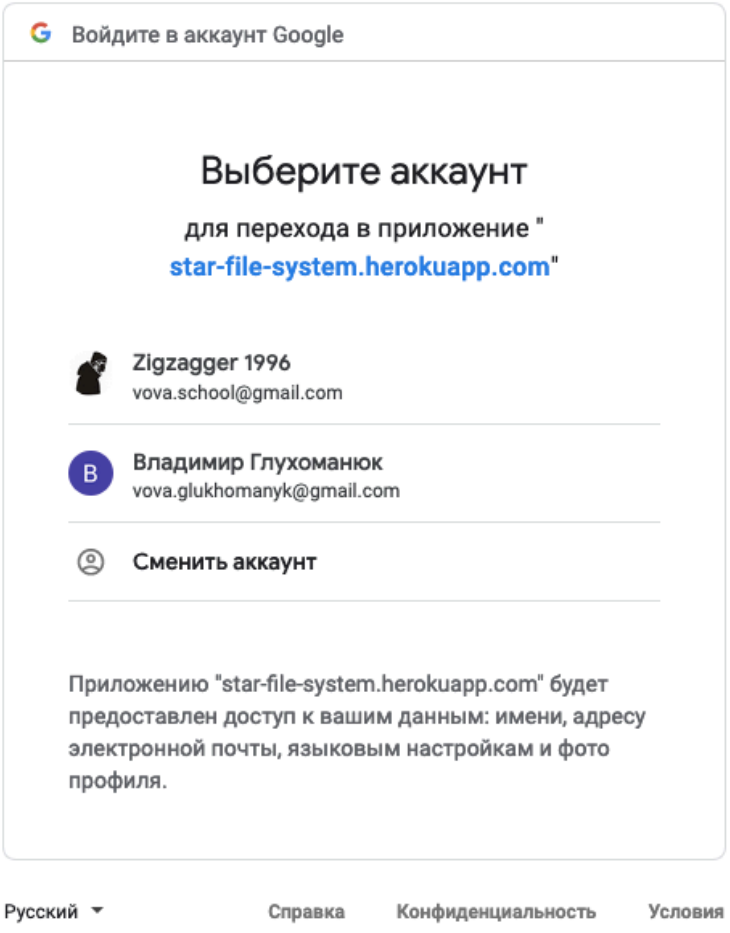

Рисунок 6.1 – Сторінка авторизації користувача

На рисунку 6.1 зображена форма авторизацію створеної за допомогою сервісів google.

Після перевірки даних, і їх коректність, користувачу буде надано доступ до хмарного сховища (рисунок 6.2).

Користувач має доступ одразу до усіх методів адміністрування, яким було створено персональну візуалізацію, для полегшення роботи з хмарним сховищем.

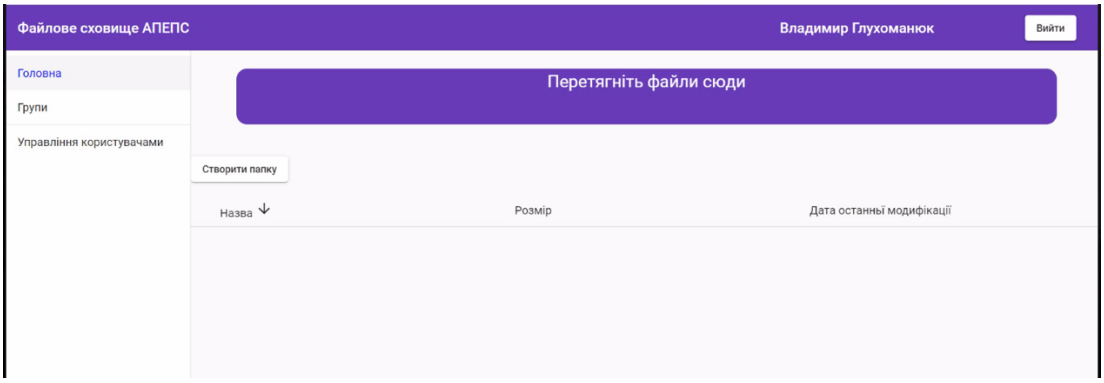

Рисунок 6.2 – Головне меню хмарного сховища

На даному рисунку зображено головне меню хмарного сховища. У якому користувач має змогу виконувати базові функції.

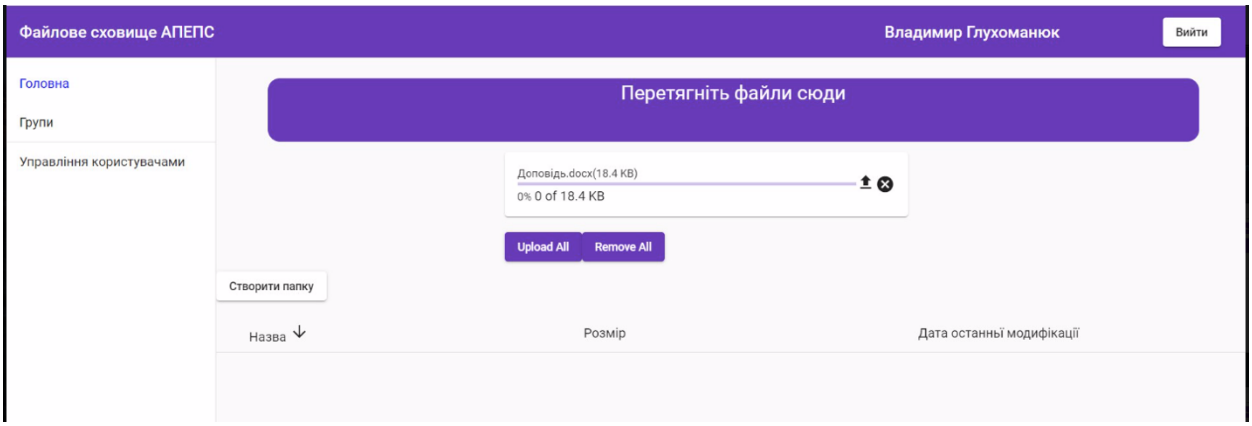

Рисунок 6.3 – Завантаження файлу у хмарне сховище

Після завантаження файлу у хмарове сховище, користувачі які мають доступ до цього файлу, будуть мати змогу закачати його на свій пристрій.

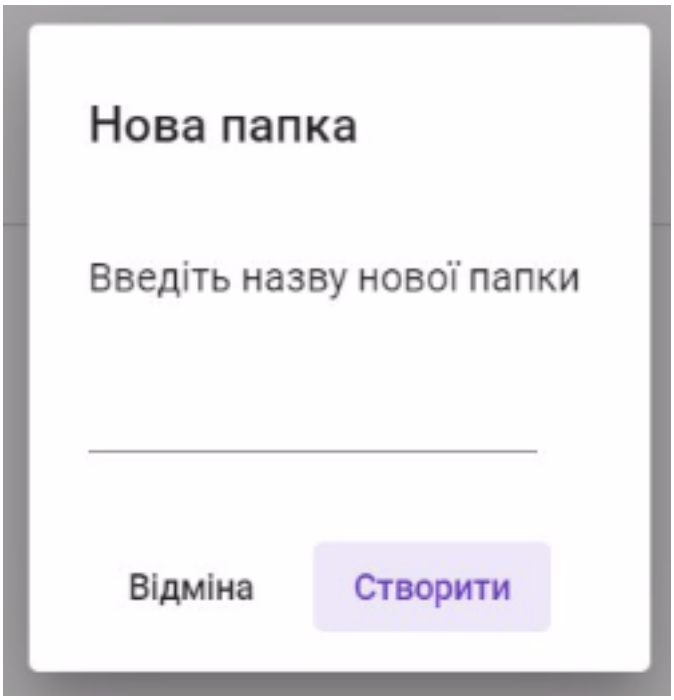

Рисунок 6.4 – Створення нової директорії

Користувач також має змогу створити нову директорію, за допомогою наданого йому графічного інтерфейсу. Ввівши попередньо назву директорії, користувач

створює у своєму сховище даних папку, з якою можуть взаємодіяти усі, хто має до неї доступ.

Висновки до розділу 6

У даному розділі описано логіку роботи системи й подано інформацію для користувачів системи з метою полегшення їхньої подальшої взаємодії із розробленими додатками. Подано зразки інтерфейсу для спрощення сприйняття інструкції.

ВИСНОВКИ

Проаналізовано технології створення засобів адміністрування хмарного середовища AWS, Google Cloud Platform, Heroku і виявлено переваги технології Heroku. Створено набір функцій, які дозволяють взаємодію користувача з хмарним середовищем, в основі якої лежить використання архітектури REST. Досліджено можливості реалізації програмного інтерфейсу написаного мовою Java, та його взаємодія з користувацьким інтерфейсом.

Реалізовано програмний інтерфейс взаємодії розробника з хмарним середовищем та інтерфейс взаємодії з користувачем. Виявлено ряд переваг при використанні архітектури, який базується на звернені до методів веб-аплікації за допомогою url. Вдалось модулі Heroku, що базуються на одній платформі, та засоби, Spring Framework що реалізують безпечну та швидку роботу з файлам, які можуть працювати на будь якій системі. Це дозволило суттєво розширити можливості розробки, модифікації, та інсталяції розробленої системи.

Виявлено переваги підходу, який передбачає передачу параметрів по http протоколу у методи веб-аплікації, адже це дозволяє взаємодіяти з системою з будь якого пристрою який підтримує браузер.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Термін – Застосунок [Електронний ресурс] – Режим доступу до ресурсу: http://www.setlab.net/?view=application-term
2. Взаємодія програмних систем [Електронний ресурс]. — Режим доступу до ресурсу: https://geektimes.com/post/282922/
3. Офіційний сайт - Heroku [Електронний ресурс] – Режим доступу до ресурсу: https://blog.heroku.com/
4. Офіційна специфікація – Веб служби [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/2002/ws/
5. Офіційна специфікація - SOAP [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/soap/
6. Офіційна специфікація – XML [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/XML/
7. Офіційна специфікація – WSDL2.0 [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/wsdl20/
8. Що таке RESTful API? [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/601
9. QAInfo [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/soap-web-servis-api-testing-interview/
10. Офіційний сайт – Apache Maven [Електронний ресурс] – Режим доступу до ресурсу: http://maven.apache.org/
11. Amazon S3 – офіційна документація [Електронний ресурс] – Режим доступу до ресурсу:

https://aws.amazon.com/ru/s3/

12. Офіційна документація – Google Cloud Platform [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/newsletter/
13. Офіційна документація – IntelliJ IDEA 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/documentation/
14. NIO2. Новый Java IO новый. [Електронний ресурс] – Режим доступу до ресурсу: http://amaloff.blogspot.com/2015/09/nio2-java-io.html
15. LinuxQuestions [Електронний ресурс] – Режим доступу до ресурсу: https://www.linuxquestions.org/reviews/index.php

Додаток А

Інструментальні засоби адміністрування кафедрального хмарного сховища

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51169_18Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ41348_18Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ41348_18Б 12-1	StarFileSystem	Основний програмний інтерфейс інструментів адміністрування

ДОДАТОК Б

Інструментальні засоби адміністрування кафедрального хмарного сховища

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51169_18Б

Аркушів 3

Київ 2019

Текст програми інструментальних засобів адміністрування кафедрального хмарного сховища

```
package ua.kpi.tef.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import ua.kpi.tef.entity.*;
import ua.kpi.tef.item.operations.ItemOperations;
import ua.kpi.tef.service.GroupService;
import ua.kpi.tef.service.ItemService;
import ua.kpi.tef.service.UserService;

import javax.xml.ws.http.HTTPException;
import java.util.*;

@RestController
@RequestMapping("/api")
public class ItemController {

    private final GroupService groupService;
    private final UserService userService;
    private final ItemService itemService;

    @Autowired
    public ItemController(GroupService groupService, UserService userService, ItemService itemService) {
        this.groupService = groupService;
        this.userService = userService;
        this.itemService = itemService;
    }

    @GetMapping("/get/{id}")
    public ItemWithPermission getItemById(@PathVariable Long id, @AuthenticationPrincipal User user) {
        user = userService.findUserByEmail(user.getEmail()).get();
        Item item = itemService.findById(id).get();
        if (item.getParent() != null && item.getParent().getUserItemPermissionMap().get(user) == null) {
            item.setParent(null);
        }
        if (user.getItems().containsKey(item)) {
            return new ItemWithPermission(item, user.getItems().get(item));
        } else {
            throw new HTTPException(403);
        }
//        if (userService.findUserByEmail(user.getEmail()).get().getItems().containsKey(itemService.findById(id).get())) {
//            return new itemService.findById(id).orElse(null);
//        } else {
//            throw new HTTPException(403);
//        }

    }

    @GetMapping(value = "/getAllItems")
    @ResponseBody
    public List<ItemWithPermission> getAllItems(@RequestParam(required = false) Long idParent, @AuthenticationPrincipal User user) {
        System.out.println("In Get All Items");
        User currentUser = userService.findUserByEmail(user.getEmail()).get();

        if (currentUser.getItems() == null) {
            return new ArrayList<>();
        }

        Map<Item, ItemPermission> items = currentUser.getItems();
        List<ItemWithPermission> itemWithPermissions = new ArrayList<>();
        Item parentItem = idParent == null ? null : itemService.findById(idParent).get();
//        Item parentItem = itemService.findById(idParent).orElse(null);
        if (parentItem == null) {
            for (Map.Entry<Item, ItemPermission> entry : items.entrySet()) {
                if (entry.getKey().getParent() == null || entry.getKey().getParent().getUserItemPermissionMap().get(currentUser) == null) {
```

```
                itemWithPermissions.add(new ItemWithPermission(entry.getKey()
                        .setUserItemPermissionMap(null)
                        .setParent(null), entry.getValue()));
                }
            }
        } else {
            for (Map.Entry<Item, ItemPermission> entry : items.entrySet()) {
                if (parentItem.equals(entry.getKey().getParent())) {
                    ItemWithPermission itemWithPermission = new ItemWithPermission(entry.getKey()
                            .setUserItemPermissionMap(null)
                            .setParent(null)
                            , entry.getValue());

                    itemWithPermissions.add(itemWithPermission);
                }
            }
        }
        return itemWithPermissions;
    }

    @PostMapping(value = "/upload", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseBody
    public ResponseEntity<String> uploadItem(@RequestParam MultipartFile file, @RequestParam(required = false) Long idParent,
                                             @AuthenticationPrincipal User user) {
        Item itemToSave
= new Item().setName(file.getOriginalFilename())
                .setLastModified(new Date().getTime())
                .setDirectory(false)
                .setSize(file.getSize());
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        System.out.println("Upload");
        if (idParent == null) {
            System.out.println("With No Parent");
            if (ItemOperations.getInstance().uploadItem(file, ItemOperations.DEFAULT_PATH + "/" + file.getOriginalFilename())) {
                itemToSave.setPath(ItemOperations.DEFAULT_PATH + "/" + file.getOriginalFilename());
                Map<User, ItemPermission> userItemPermissionMap = new HashMap<>();
                userItemPermissionMap.put(currentUser, ItemPermission.OWNER);
                itemToSave.setUserItemPermissionMap(userItemPermissionMap);
                itemService.create(itemToSave);
                return new ResponseEntity<>("Uploaded", HttpStatus.ACCEPTED);
            } else return new ResponseEntity<>("Not Uploaded", HttpStatus.BAD_REQUEST);
        }

        System.out.println("With Parrent");
        Item item = itemService.findById(idParent).get();
        if (!ItemPermission.OWNER.equals(currentUser.getItems().get(item))) {
            return new ResponseEntity<>("You can't upload", HttpStatus.BAD_REQUEST);
        }
//        itemToSave.setParent(item);
        if (ItemOperations.getInstance().uploadItem(file, item.getPath() + "/" + file.getOriginalFilename())) {
            itemToSave.setParent(item).setPath(item.getPath() + "/" + file.getOriginalFilename());
            Map<User, ItemPermission> parentUserPermissions = new HashMap<>();
            parentUserPermissions.putAll(item.getUserItemPermissionMap());
            itemToSave.setUserItemPermissionMap(parentUserPermissions);
            itemService.create(itemToSave);
            return new ResponseEntity<>("Uploaded", HttpStatus.ACCEPTED);
        } else return new ResponseEntity<>("Not Uploaded", HttpStatus.BAD_REQUEST);


    }

    @GetMapping(value = "/download/{id}")
    @ResponseBody
    public ResponseEntity downloadItem(@PathVariable Long id, @AuthenticationPrincipal User user) {
        Item item = itemService.findById(id).get();
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        if (item.getUserItemPermissionMap().containsKey(currentUser)) {
            return ItemOperations.getInstance().downloadItem(item.getPath());
        }

        return new ResponseEntity<>("Can't download", HttpStatus.BAD_REQUEST);
    }

    @PostMapping(value = "/createItem")
    @ResponseBody
    public ResponseEntity<String> createItem(@RequestParam(required = false) Long idParent, String name, @AuthenticationPrincipal User user) {
```

```
        Item item = new Item().setDirectory(true)
//                .setParent(itemService.findById(idParent).orElse(null))
                .setParent(idParent == null ? null : itemService.findById(idParent).get())
                .setLastModified(new Date().getTime())
                .setName(name);
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        if (itemService.findByName(name) != null) {
            new ResponseEntity<>("Such name is occupied ", HttpStatus.BAD_REQUEST);
        }
        if (idParent != null) {
            Item itemParent = itemService.findById(idParent).get();
            if (!ItemPermission.OWNER.equals(currentUser.getItems().get(itemParent))) {
                return new ResponseEntity<>("Tou can't create", HttpStatus.BAD_REQUEST);
            }
            item.setPath(itemParent.getPath() + "/" + name);
            Map<User, ItemPermission> parentUserPermissions = new HashMap<>();
            parentUserPermissions.putAll(itemParent.getUserItemPermissionMap());
            item.setUserItemPermissionMap(parentUserPermissions);
            ItemOperations.getInstance().createItem(itemParent.getPath() + "/" + name);
            item.setParent(itemParent);
            itemService.create(item);
            return new ResponseEntity<>("Created successfully", HttpStatus.ACCEPTED);
        }
        Map<User, ItemPermission> userItemPermissionMap = new HashMap<>();
        userItemPermissionMap.put(currentUser, ItemPermission.OWNER);
        item.setUserItemPermissionMap(userItemPermissionMap);
        item.setPath(ItemOperations.DEFAULT_PATH + "/" + name);
        itemService.create(item);
        ItemOperations.getInstance().createItem(ItemOperations.DEFAULT_PATH + "/" + name);
        return new ResponseEntity<>("Created successfully", HttpStatus.ACCEPTED);
    }

    @DeleteMapping(value = "/deleteItem/{id}")
    @ResponseBody
    public ResponseEntity<String> deleteItem(@PathVariable Long id, @AuthenticationPrincipal User user) {
        Item item = itemService.findById(id).get();
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        if (!ItemPermission.OWNER.equals(currentUser.getItems().get(item))) {
            return new ResponseEntity<>("You can't delete", HttpStatus.BAD_REQUEST);
        }
        itemService.delete(id);
        ItemOperations.getInstance().deleteItem(item.getPath());

        return new ResponseEntity<>("Deleted successfully", HttpStatus.ACCEPTED);
    }

    @PutMapping(value = "/share/{id}")
    @ResponseBody
    public ResponseEntity<String> shareItem(@PathVariable Long id, String email, ItemPermission itemPermission,
                                            @AuthenticationPrincipal User user) {
        if (!userService.findUserByEmail(email).isPresent()) {
            return new ResponseEntity<>("Such user doesn't exists", HttpStatus.BAD_REQUEST);
        }
        User userToShare = userService.findUserByEmail(email).get();
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        Item itemToShare = itemService.findById(id).get();
        List<Item> itemList = itemService.getChildTree(itemToShare);
        if (!ItemPermission.OWNER.equals(currentUser.getItems().get(itemToShare))) {
            return new ResponseEntity<>("You cant share this", HttpStatus.BAD_REQUEST);
        }
        for (Item item : itemList) {

            item.getUserItemPermissionMap().put(userToShare, itemPermission);

        }
        itemToShare.getUserItemPermissionMap().put(userToShare, itemPermission);
        itemService.create(itemToShare);
        return new ResponseEntity<>("Item Shared", HttpStatus.ACCEPTED);
    }

    @PutMapping(value = "/share_group/{id}")
    @ResponseBody
    public ResponseEntity<String> shareItemWithGroup(@PathVariable Long id, Long idOfGroup, ItemPermission itemPermission,
                                                      @AuthenticationPrincipal User user) {
        if (!groupService.findGroupById(idOfGroup).isPresent()) {
            return new ResponseEntity<>("Such group doesn't exists", HttpStatus.BAD_REQUEST);
        }
```

```
        Group groupToShare = groupService.findGroupById(idOfGroup).get();
        Set<User> usersToShare = groupToShare.getUsersWithPermissions().keySet();
        usersToShare.removeIf(user1 -> user1.getEmail().equals(user.getEmail()));
        usersToShare.forEach(userToShare -> shareItem(id, userToShare.getEmail(), itemPermission, user));
        return new ResponseEntity<>("Item Shared", HttpStatus.ACCEPTED);
    }

    @GetMapping(value = "/users/{id}")
    @ResponseBody
    public List<UserWithItemPermission> users(@PathVariable Long id, @AuthenticationPrincipal User user) {


        Optional<Item> item = itemService.findById(id);
        if (!item.isPresent()) {
            return new ArrayList<>();
        }
        user = userService.findUserByEmail(user.getEmail()).get();
        if (!user.getItems().containsKey(item.get()) || !ItemPermission.OWNER.equals(user.getItems().get(item.get()))) {
            return new ArrayList<>();
        }

        List<UserWithItemPermission> userWithPermissions = new ArrayList<>();
        item.get().getUserItemPermissionMap().forEach((key, value) -> userWithPermissions.add(new UserWithItemPermission(key, value)));
        return userWithPermissions;
    }

    @DeleteMapping(value = "/removeAccess/{id}")
    @ResponseBody
    public ResponseEntity removePermission(@PathVariable Long id, @RequestParam String email, @AuthenticationPrincipal User currentUser) {
        if (!itemService.findById(id).isPresent() || !userService.findUserByEmail(email).isPresent()) {
            return new ResponseEntity(HttpStatus.BAD_REQUEST);
        }
        currentUser = userService.findUserByEmail(currentUser.getEmail()).get();


        User user = userService.findUserByEmail(email).get();
        Item itemToRevertAccess = itemService.findById(id).get();
        List<Item> itemList = itemService.getChildTree(itemToRevertAccess);
        for (Item item : itemList) {
            item.getUserItemPermissionMap().remove(user);
            itemService.create(item);
        }
        itemToRevertAccess.getUserItemPermissionMap().remove(user);
        itemService.create(itemToRevertAccess);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Текст методів реалізацій програмного інтерфейсу

```
package ua.kpi.tef.item.operations;

import org.springframework.core.io.InputStreamResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.multipart.MultipartFile;
import ua.kpi.tef.entity.Item;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;

public class ItemOperations implements ItemCommand {

    private static ItemOperations itemOp = null;
```

```
private File rootDirectory;
public static final String DEFAULT_PATH = "./root";

private ItemOperations() throws IOException {

    rootDirectory = new File(DEFAULT_PATH);
    rootDirectory.delete();
    Path dirPathObj = Paths.get(DEFAULT_PATH);
    if (Files.exists(dirPathObj)) {
        deleteItem(DEFAULT_PATH);
    }
    Files.createDirectories(dirPathObj);

}

public static ItemOperations getInstance() {
    if (itemOp == null) {
        try {
            itemOp = new ItemOperations();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return itemOp;
    } else {
        return itemOp;
    }
}

@Override
public List<Item> getAllItems(String path) {
    File directory = new File(path);
    if (directory.listFiles() == null) {
        return null;
    } else {
        List<Item> listOfFiles = new ArrayList<>();
        for (File f : Objects.requireNonNull(directory.listFiles())) {
            listOfFiles.add(new Item(f));
        }
        return listOfFiles;
    }
}

@Override
public String createItem(String path) {
    Path dirPathObj = Paths.get(path);
    boolean dirExists = Files.exists(dirPathObj);
    if (dirExists) {
        return "Directory Already Exists !";
    } else {
        try {
            // Creating The New Directory Structure
            Files.createDirectories(dirPathObj);
            System.out.println("! New Directory Successfully Created !");
            return dirPathObj.toString() + "is created";
        } catch (IOException ioExceptionObj) {
            return "Problem Occured While Creating The Directory Structure= " + ioExceptionObj.getMessage();
        }
    }
}

@Override
public String deleteItem(String path) {
    File item = new File(path);
    if (item.isDirectory()) {
        try {
            Files.walk(item.toPath())
                    .sorted(Comparator.reverseOrder())
                    .map(Path::toFile)
                    .forEach(File::delete);
            return item.getName() + "is deleted";
        } catch (IOException e) {
            e.printStackTrace();
            return item.getName() + "is not deleted";
        }
    } else {
        item.delete();
        return item.getName() + "is deleted";
    }
```

```
    }

    @Override
    public ResponseEntity<Object> downloadItem(String path) {
        try {
            File file = new File(path);

            InputStreamResource resource = new InputStreamResource(new FileInputStream(file));
            HttpHeaders headers = new HttpHeaders();
            headers.add("Content-Disposition", String.format("attachment; filename=\"%s\"", file.getName()));
            headers.add("Cache-Control", "no-cache, no-store, must-revalidate");
            headers.add("Pragma", "no-cache");
            headers.add("Expires", "0");

            return ResponseEntity.ok().headers(headers).contentLength(file.length()).contentType(MediaType.parseMediaType("application/txt")).body(resource);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>("error occurred", HttpStatus.BAD_REQUEST);
        }
    }

    @Override
    public boolean uploadItem(MultipartFile file, String path) {

        try {
            Path filePath = Paths.get(path);

            Files.copy(file.getInputStream(), filePath, StandardCopyOption.REPLACE_EXISTING);
            return true;
        } catch (IOException ex) {
            return false;
        }
//        try {
//
//            File convertFile = new File(path);
//            convertFile.createNewFile();
//            FileOutputStream fout = new FileOutputStream(convertFile);
//            fout.write(file.getBytes());
//            fout.close();
//            return new ResponseEntity<>("File is uploaded successfully", HttpStatus.OK);
//        } catch (IOException ex) {
//            ex.printStackTrace();
//            return new ResponseEntity<>("File not uploaded", HttpStatus.EXPECTATION_FAILED);
//        }
    }
}
```

ДОДАТОК В

Інструментальні засоби адміністрування кафедрального хмарного сховища

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51169_18Б

Аркушів 9

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис інструментальних засобів адміністрування кафедрального хмарного сховища. Створені інструменти виконують такі завдання:

— завантаження та закачування файлів;

— операції з директоріями;

— надання обмеження доступу;

— виведення результатів роботи методів.

Програм отримує дані через url та оброблює їх а віддаленому сервері.

При розробці програми використовувалась мова Java у наступному середовищі програмування: IntelliJ IDEA, із використанням Spring Framework, та архітектури REST.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис інструментів взаємодії користувача з хмарним середовищем. У додатку Б міститься програмний код головних класів розроблюваної програмної системи.

Розроблена система працює в будь якій операційній системі і потребує встановленого на пристрій останньої версії Java.

При розробці веб-аплікації використовувалась мова Java з використанням середовища:

1. IntelliJ IDEA 2019.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний інтерфейс виконує завдання надання розробнику гнучкого інструменту для реалізації хмарного сховища, а саме надає базовий набір інструментів для роботи з файловою системою та користувачами хмарного сховища.

Також розроблені додатки можуть використовуватися в якості учбових матеріалів при опрацюванні технологій хмарних сховищ.

Функціональні обмеження на використання інтерфейсу полягають лише в кількості створених інструментів взаємодії.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації інструментів адміністрування хмарного сховища знадобилось створення програмного інтерфейсу на базі REST архітектури та Spring Framework. Одним з інструментів для створення взаємодії між усіма компонентами програми є Apache Maven.

В Java було виконано створення гнучкого API. В IntelliJ IDEA було використано спеціальний тип проекту під назвою “Maven Project ”, результатом чого стало створення гнучкого програмного забезпечення.

При запуску хмарного середовища спочатку пройти авторизацію, далі данні перевіряються, та надається можливість використання хмарного сховища даних.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої ефективності роботи створеного хмарного середовища було обрано IntelliJ IDEA 2019 яка показала себе надійною та гнучкою середою розробки програм.

Розроблена система працює в будь якій операційній системі Windows 7, Windows 8 та Windows10, MacOS, Linux і потребує встановленого на ПК пакету програм Intellij IDEA 2019 та JDK.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблені системи не потребують інсталяції, достатньо запустити веб сайт та пройти авторизацію.

Після запуску користувач отримає доступ до хмарного сховища та інструментів адміністрування.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розробленого інтерфейсу є інформація яка зчитується з url. Вхідні дані можуть мати вигляд цілого числа або числа з плаваючою комою, або файлом, чи словом. Вихідними даними результат роботи методу, наприклад список файлів у директорії, чи створений файл.